Title

A New Framework for Loop-Free On-Demand Routing Using Destination Sequence Numbers

Authors

Garcia-Luna-Aceves, J.J.
Rangarajan, H.

Publication Date

2004-10-24

Peer reviewed

A New Framework for Loop-Free On-Demand Routing Using Destination Sequence Numbers *

J.J. Garcia-Luna-Aceves Hari Rangarajan
Department of Computer Engineering
University of California
Santa Cruz, CA 95064, U.S.A.
Email: {jj, hari}@cse.ucsc.edu

Abstract

A generalized framework for loop-free routing based entirely on destination sequence numbers is presented. The framework eliminates the counting-to-infinity problem found in AODV and other on-demand routing protocols based on destination sequence numbers. The Sequence-Number Window Routing (SWR) protocol is presented as an example of this framework. SWR is compared via simulations with DSR, AODV and OLSR using networks of 50 and 100 mobile nodes; the results indicate that SWR is as efficient as AODV, without incurring counting to infinity.

1 Introduction

Several routing protocols have been proposed to date for wireless networks. Pro-active protocols for MANET like the Optimized Link State Routing (OLSR) [1] maintain routes to every possible destination in the network, and can incur temporary loops. On-demand protocols like the Adhoc On-demand Distance Vector (AODV) [8], Dynamic Source Routing (DSR) [4], and the Labeled Distance Routing (LDR) protocol [2] establish routes to only those destinations for which there is traffic, and attempt to ensure loop freedom at every instant to limit control overhead. To avoid routing table loops, on-demand routing protocols use source routing, sequence numbers, or nodal synchronization.

DSR establishes a loop-free route to a destination by carrying the path traversed in the route request and the reverse path is then used to source route data packets. On a link failure, reliable error updates have to be sent to the source, so that a new route can be searched. Furthermore, we have recently shown [6] how path information can be used to attain loop-free routing, without the need for the header of a data packet to specify a source route.

AODV maintains loop freedom with the use of per-destination sequence numbers. The sequence number carried in a route request elicits route replies with an equal or higher sequence number. On a link failure, a node increases its sequence number for a destination and invalidates the route. The key limitation with AODV's approach to destination sequence numbers is that it prevents responses from nodes that are closer to the destination but have an older sequence number, even if they have a valid loop-free path to the destination. Consequently, the likelihood that the destination itself must resolve a route request is very high, because the destination is the only node that can increase its own sequence number. LDR [2] overcomes this limitation of AODV by using distances as an additional invariant. Nodes towards the destination are ordered by their shortest known distance. The destination sequence number is used simply as a "reset" of such distances. Although LDR performs much better than AODV [2], it requires the addition of an extra invariant to the destination sequence number.

We present a new framework for the development of routing protocols that can attain loop freedom safely based solely on destination sequence numbers, even when nodes are allowed to delete such sequence numbers at any time. Section 2 shows how deleting destination sequence numbers can lead to counting-to-infinity behavior in such protocols as AODV and LDR. Section 3 describes a new framework for on-demand routing using solely destination-based sequence numbers. The novelty of our framework is that it treats destination sequence numbers as a finite label space, rather than the traditional notion of absolute timestamps. Another contribution in this work is the introduction of *sequence number window*s as a technique for realizing the progressive ordering of sequence numbers when establishing paths to a destination. This allows more intermediate nodes to resolve route requests, given that nodes closer to the destination have higher sequence numbers than nodes

*This work was funded in part by the Baskin Chair of Computer Engineering at UCSC.

farther away. Section 4 introduces the Sequence-number Window Routing (SWR) protocol as an example of the new framework. Nodes in SWR adopt sequence numbers for a destination in a non-decreasing order along each path to the destination within a window of sequence numbers whose range is controlled by the destination. To handle node failures, reboots, and nodes "forgetting" about destination sequence numbers in a safe manner, SWR forces the destination to reply to a route request relayed or originated by a node that has no route entry for the destination. In such a case, the destination issues a reply incrementing its own sequence number to a value greater than the upper bound of the previous sequence-number window. Each node propagating a valid route reply adopts a sequence number within the new window in a way that sequence numbers are always non-decreasing along any path to the destination. This allows destination sequence numbers within a window to be recycled along paths, before the destination is forced to resolve a route request by incrementing its sequence number again. Section 5 provides an example of how SWR operates. Section 6 analyzes the correctness of SWR. Section 7 compares the performance of SWR against two on-demand protocols (AODV, DSR) and a proactive link state protocol (OLSR). Section 8 provides our concluding remarks.

2 Counting to Infinity in AODV

In the rest of this paper, sn_D^A denotes the sequence number stored at node A for destination D, d_D^A denotes the distance from node A to destination D, lc_B^A denotes the cost of the link from node A to neighbor B, and rt_D^A denotes the state of the routing-table entry for destination D at node A (either null (ϕ), valid or invalid).

The AODV RFC [8] recommends that nodes delete invalid route entries after a finite time equal to the maximum elapsed time after which a node can still send data packets to the next-hop specified in the routing table, called the $DELETE\_PERIOD$. However, as we show, the condition is not safe.

A directed acyclic successor graph is shown in Fig. 1(a) for destination D. We assume that all nodes run AODV correctly and have data for D at some point in time. The dotted node R between node C and node X is used to indicate a set of one or more nodes that could be along the path. All nodes are assumed to have sn_D^1 as the destination sequence number for D in their routing tables. We consider path P= $\{Y, X, \{R\}, C, B, A\}$ and we trace one of many sequences of events that can cause counting to infinity for destination D. Assume that link $e1$ fails, which results in node D being isolated from the connected component consisting of nodes A,B,C,$\{R\}$,X and Y. Now A detects the unreachability of D within a finite time through either a link-layer notification or HELLO messages. Node A invalidates the route to D, increments its sequence number for destination D to sn_D^2 (hence, $sn_D^2 > sn_D^1$), and sends a route error (RERR) to node B, which may not be delivered. This sequence of events is represented in Fig. 1(b).

After the above sequence of events, node A searches for a new route to D with sn_D^2 as the required sequence number. However, destination D is unreachable and none of the other nodes can satisfy the request, because their sequence numbers equal sn_D^1. In our scenario, node B eventually invalidates its route entry for destination D, given that node A sends RERRs to B every time it receives a data packet for the invalid route to D.

Let t_1 be the time after which node A receives no data packets from B, node A deletes the invalid route for D with sn_D^2 at time $t_{del_D^A} = t_1 + DELETE\_PERIOD$. Node B invalidates its route entry for destination D at a time $t > t_1$, and notifies C of the unreachability of D proceeding in the same fashion as the RERR exchange between A and B. This results in C invalidating its routing entry for D. The nodes along path P learn about the unreachability of node D as the RERRs are propagated.

Let t_y be the time when Y invalidates its routing entry for D. At any time $t' < t_y$, any route search for destination D with a sequence number $sn_D > sn_D^1$ cannot be answered by any node in the network.

At a time $t_{req_D^A} > t_{del_D^A}$, node A sends a RREQ with an invalid sequence number for D. If at time $t_{rep_D^Y} < t_y$ node Y receives the RREQ, then it sends a RREP for D with a sequence number sn_D^1, which can now be used by node A to create a routing entry for destination D with sn_D^1. This results in the formation of an undirected cycle, which is shown in Fig. 1(c). Similarly, once the $DELETE\_PERIOD$ elapses, node B deletes its invalid entry for D after attempting to find a route with sequence number sn_D^2. A RREQ for D by node B with an invalid sequence number can then be answered by node A with sn_D^1. This effect cascades to other nodes along the path, and counting to infinity occurs in this connected component. The route lifetimes at each node can be kept alive by the constant flow of data packets for D that either originate locally at the node or are forwarded along the undirected cycle.

The above is an example of a basic problem that can be summarized as follows: *A node A along a path P to destination D should never delete its invalid route table entry for D before guaranteeing that all its upstream nodes along path P have invalidated their active route entries for D.*

Of course, temporary looping can occur in AODV and any other protocol using the same destination-based sequence numbering approach, given that counting-to-infinity can happen. In practice, counting to infinity in AODV can be avoided by waiting "long enough" before deleting invalid routes or rejoining normal operation after reboots. However, as the network size and its diameter change, what

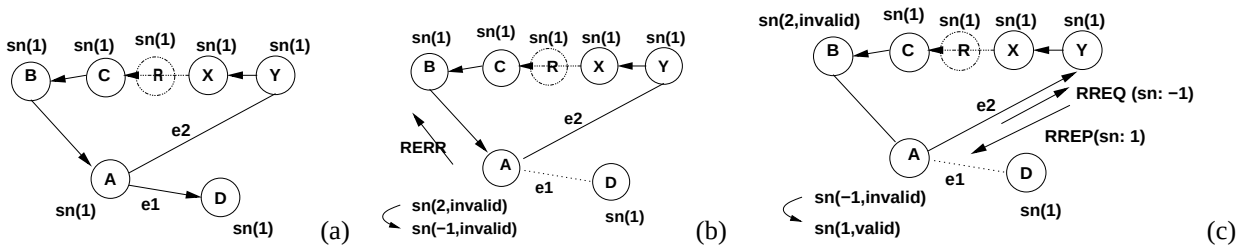


Figure 1. AODV Count-to-infinity example

"long enough" means must also change. Given that internodal coordination spanning multiple hops incurs too much overhead and that very long waiting periods are undesirable for protocol efficiency, a more elegant solution to the counting-to-infinity problem is desirable, which we present in the next section.

3 Framework for Loop-Free On-demand Routing Using Sequence Numbers

In the past, loop-free routing approaches based on destination sequence numbers have relied on the premise that a sequence number serves the purpose of time-stamping an update, and thus accepting the update for a destination with the latest sequence number maintains loop-freedom. Adopting this approach, the following sufficient condition for loop-free routing using destination sequence numbers has been used in AODV and other protocols in the past [8], [7].

Sequence Number Condition (SNC): Node A can make node B its successor for destination D after processing an input event if $(sn_D^A < sn_D^B)$ or $(sn_D^A = sn_D^B \wedge d_D^A > d_D^B)$. If no neighbor satisfies one of the above conditions, then node A must keep its current successor if it has any.

The proof that SNC can be used to enforce loop freedom is presented in [7] under the implicit assumption that nodes never "forget" the last sequence numbers they learn for a given destination.

Rather than considering sequence numbers as time-stamps, we model the sequence numbers of a destination as a finite label space from $[0, \ldots, 2^{n-1}]$ in which n is the number of bits allocated for storing the sequence number. By doing so, the problem of maintaining loop-freedom using sequence numbers reduces to a case of assigning destination sequence numbers as labels for a destination at each node along the successor path. Following this approach, we augment SNC with the following new condition that allows nodes to label themselves with a sequence number for a destination without creating loops.

Sequence Label Condition (SLC): Let PS_D^A denote the set of neighbors of A that use it as next hop to D. When A makes B its successor for destination D after processing an input event that satisfies $sn_D^A < sn_D^B$, node A can assign (label) itself a destination sequence number that satisfies $\text{MAX}(sn_D^j) < sn_D^A \leq sn_D^B, \ \forall j \in PS_D^A$.

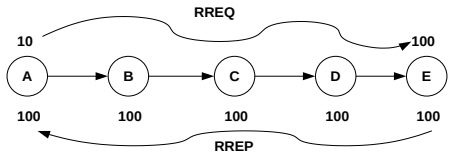
Theorem 1 *Using SNC for choosing successors and SLC for updating sequence numbers at nodes for destination D cannot create loops if no node ever forgets the largest sequence number it learns for a destination.*

Proof: The proof follows from the fact that SNC is sufficient to enforce loop freedom by showing that $sn_D^{n_i} \leq sn_D^{n_{i-1}}$ for $i \in \{2, k\}$ along any successor path $P = \{n_k, \ldots, n_1\}$.

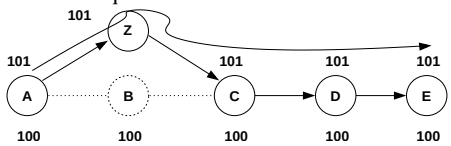
For path P to exist at a given time t, it must be true that all nodes in P have a successor. According to SNC, node n_i can make n_{i-1} as its successor for destination D if $sn_D^{n_i} < sn_D^{n_{i-1}}$ or $sn_D^{n_i} = sn_D^{n_{i-1}} \ \wedge \ d_D^{n_i} < d_D^{n_{i-1}}$. Consider first the case in which $sn_D^{n_i} < sn_D^{n_{i-1}}$. If node n_D^{i+1} uses n_D^i as its successor to destination D when n_i makes n_{i-1} its successor for the same destination, then it follows from SNC that $sn_D^{n_{i+1}} \leq sn_D^{n_i}$. Hence,node n_i can set $sn_D^{n_{i+1}} < sn_D^{n_i} \leq sn_D^{n_{i-1}}$ according to SLC, which does not violate the ordering of sequence numbers along path P. In the second case, if node n_i sets $sn_D^{n_i} = sn_D^{n_{i-1}}$, the sequence-number ordering is not affected either. ■

As stated, SLC allows nodes to assign themselves a destination sequence number *smaller* than the latest destination sequence number received in an update. However, this requires synchronization with neighbor nodes to determine the current set of predecessors (neighbors using the node as sucessor) and their destination sequence numbers. Unfortunately, this synchronization incurs additional signaling.

To avoid any synchronization, a node can update itself to the latest destination sequence number reported in a control message, which is the approach adopted in AODV. However, this results in the destination node being the only one that can resolve most RREQs. For example, Fig. 2 (a) shows a five-node network topology. Node A has an invalid route to E with a sequence number $sn_E^A = 10$. Destination E has a sequence number $sn_E^E = 100$. Node A trying to establish a route to E sends a route request (RREQ) that is answered by the destination E, and the route reply (RREP) carries $sn_E^{rep} = 100$. Nodes B,C, and D along A's successor path update their destination sequence numbers for E to

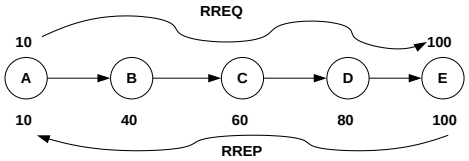


(a) AODV - Sequence numbers as absolute time stamps

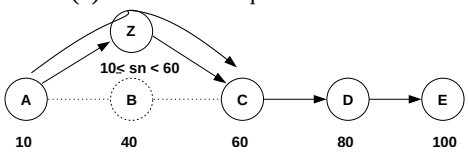


(b) AODV - On Route Failure

Figure 2. Sequence number assignment in AODV



(a) Window of Sequence numbers



(b) Fast recovery on route failure

Figure 3. Sequence number Windows

100. Now, if at a later time if node B crashes and an alternate path through node Z exists as shown in Fig. 2 (b). Node A attempts to re-establish its route to E. Node A's RREQ carries a increased sequence number $sn_E^{req} = 101$, which prevents any of the nodes along the path ZCD from replying, even though nodes C and D have a loop-free active route. Eventually, the request reaches the destination E, which now responds with a increased sequence number $sn_E^E = 101$ and the successor path ZCD from A to E is established.

By allowing a *progressive* ordering of the sequence numbers, it is possible to expedite route recovery and reduce control overhead. We introduce the *Sequence Number Window* (SNW) as a tool to achieve this type of ordering. A sequence number window along any path to a destination D is formed by a set of nodes $\{n_k, \ldots, n_1\}$ such that $sn_D^{n_k} < sn_D^{n_1}$ and $(sn_D^{n^i} \leq sn_D^{n_k} \wedge sn_D^{n_i} < sn_D^{n^1}), \forall i \in \{k-1, 2\}$.

Fig. 3 (a) illustrates a sequence number window between node A and E ranging from $[10, ..., 100]$ assuming a initial configuration where nodes B,C, and D have no route for E. Nodes label themselves with sequence numbers less than the latest known sequence number, which amounts to distributing the sequence numbers inside the window while maintaining the ordering. Fig. 3 (b) illustrates the benefits of distributing sequence numbers inside a window compared to AODV's approach. As in the previous example, node B fails, and node A obtains a response from node C when it attempts to obtain a new path to E, because A's increased sequence number in the request $sn_E^{req} = 11$ can be satisfied by node C. With on-demand routing protocols that resort to expanding ring searches, this scheme enables more nodes with active routes to E to respond.

In large MANETs, it is possible for nodes to forget previously known sequence numbers for a destination (e.g., after rebooting or by deleting old invalid routing-table entries). From Theorem 1, it follows that looping can occur only if nodes adopt new successors without knowing a prior sequence number for the destination. To ensure the safe use of an SNW or destination-based sequence numbers, our framework requires that *any RREQ relayed or originated by a node without a valid sequence number for the destination be answered by the destination*. The example shown in Fig. 3 with the added constraint for safety remains unchanged if node Z has a destination sequence number for E lying in the range ($10 \leq sn < 60$); however, if Z has no state for E, then the request must be answered only by the destination.

Based on the above approach to destination sequence numbering, the following rules for loop-free on-demand routing can be defined based on the facts that (a) each node stores a sequence number for every active destination, and (b) nodes invalidating their route to a destination retain the sequence number. We assume a generic control message framework consisting of route request (RREQ), route reply (RREP), and route error (RERR) messages similar to that of other on-demand routing protocols. The routing-table entry at node A for destination D includes the current sequence number (sn_D^A), the current route cost (d_D^A), and the successor (s_D^A). If no routing entry exists at node A for destination D, then the current sequence number is considered unknown (i.e., $sn_D^A = -1$). The superscripts req and rep are used for variables included in a RREQ and RREP, respectively. The parameter $reset$ denotes, if the RREQ requires the destination to reply or if the RREP was originated by the destination.

*A*SC: (Accept Sequence number Condition). When node A receives a RREP from node B for destination D, then node A sets $s_D^A \leftarrow B$ if $(sn_D^A < sn_D^{rep})$ or $(sn_D^A = sn_D^{rep}) \wedge (d_D^A > d_D^{rep})$. If $sn_D^A = -1$, then node A should accept a RREP only if $reset_D^{rep} = 1$ (i.e., generated by the destination).

*S*SC: (Start Sequence number Condition). Node I can issue a RREP responding to a RREQ for destination D if I has an active route to D, $sn_D^I > sn_D^{req}$, and $reset_D^{req} = 0$. If node $I = D$, then it must set $reset_D^{rep} = 1$, if $reset_D^{req} = 1$.

*M*SC: (Maximum Sequence number Condition). If node A relays a RREP for destination D, it sets $sn_D^{rep} \leftarrow sn_D^A$.

The relayed RREP must not change the value of $reset_D^{rep}$. Node A relays a RREQ for destination D only if A has not previously processed this RREQ and sets msn_D^{req} = $max\{msn_D^{req}, sn_D^A\}$. If $sn_D^A = -1$ or $reset_D^{req} = 1$; then, in the relayed request, it sets $reset_D^{req} = 1$.

*U*SC: (Update Sequence number Condition). If node A must change s_D^A, then it sets $d_D^A \leftarrow \infty$ and node A sends a RREQ carrying sn_D^A.

*R*SC: (Reset Sequence number Condition). If node A has no route entry for D (i.e., $sn_D^A = -1$), then a RREQ originated must have $reset_D^{req} = 1$.

A RREQ travels a loop-free path and is relayed (MSC) with the maximum of the destination sequence number of the nodes. The RREQ can be answered by a node that has a higher sequence number than the maximum sequence number carried in the RREQ (SSC) or by the destination if the RREQ was originated or relayed by a node that had no sequence number for D (RSC). After a route failure, a node attempts to reestablish a new path (USC). USC allows the destination to be the only node that can modify its sequence number. Nodes invalidating their routes on receiving a RERR are not required to update to the destination sequence number in the RERR. RSC is a safety condition that handles nodes that cannot be ordered on the basis of sequence numbers; and MSC as a special case handles nodes relaying RREQs with no sequence number state. With the above safety conditions, route entries storing destination sequence numbers can be purged safely, without causing any loops.

USC and SSC differ from AODV, which forces nodes invalidating their routes to increase the destination sequence number, and nodes reply to a route request only if they have a higher sequence number.

4 SWR

SWR augments the rules of our basic framework with a condition for detecting the boundary of windows.

*W*BC: (Window Boundary Condition). If node A receives a RREQ for destination D and $sn_D^A > sn_D^{req}$ then node A sets the window count $wc \leftarrow 1$, otherwise sets $wc \leftarrow wc + 1$. Node A caches wc for this RREQ and relays a RREQ with the cached wc.

WBC allows nodes relaying a RREQ to determine distributedly the start and end of sequence number windows, and the number of hops spanned by the window. A window count of one indicates the beginning of a new window and signals the end of a previous sequence number window (except at the source, where the window can only begin). The window count indicates the number of nodes over which the current sequence number window is being built.

4.1 Information Stored and Exchanged

The routing table at node A maintains the following parameters for every destination D: the successor (s_D^A), sequence number (sn_D^A), the distance (hop count) to the destination (d_D^A), lifetime of route, and the state of route entry (rt_D^A). If no entry for destination D exists, then it is considered equivalent to an unknown sequence number ($sn_D^A = -1$).

A RREQ consists of the tuple $\{dst,\ src,\ rreqid,\ msn_{dst}, wc_{dst}, flags\}$. The field src denotes the identifier of the source that is seeking a path to the destination (dst), $rreqid$ along with the source (src) represents a unique identifier for a RREQ generated for a destination, msn_{dst} is the maximum of the destination sequence numbers along the path traversed by this RREQ, $flags$ carries control bits, and wc_{dst} is the window count used to infer about the current sequence number window. One control bit used is the *'reset'* bit that is set when the RREQ must be answered only by the destination.

A RREP consists of the tuple $\{dst, sn_{dst},\ src, rreqid,\ d_{dst}, ttl, flags\}$. The field ttl states the lifetime of the route at the node relaying the RREP, $rreqid$ is carried in the RREP to forward it along the reverse path to the source using information cached for the RREQ $(src, rreqid)$, sn_{dst} is the destination sequence number stored at the relaying hop, d_{dst} is the distance to the destination at the relaying hop, and $flags$ contains the *'reset'* bit, which may be set if the destination originates the RREP.

The RERR is the tuple $\{orig, unreachdests\}$, where $orig$ denotes the node originating the route errors, and $unreachdests$ is the list of destinations that are not reachable at $orig$.

A node relaying a RREQ caches the tuple $\{msn_{dst},\ wc_{dst}, revHop, reset_{dst}\}$, where $revHop$ is the identifier for the node that sent the request, and is used to relay a RREP received for this $(src, rreqid)$ pair along the reverse path. Cached entries are maintained for a period of time that is long enough, so that all RREPs for the RREQ $(src, rreqid)$ will be received with high probability.

4.2 Initiating a RREQ

Node A is said to be *active* in a route computation for destination D (i.e., the RREQ) when it initiates a RREQ for the destination, and the RREQ is uniquely identified by the pair (A, ID_A). A node relaying a RREQ (A, ID_A) originated by another node is said to be *engaged* in the RREQ. A node that is not active or engaged in a route computation for destination D is said to be *passive* for that destination.

At any given time, a node can be the origin of at most one RREQ for the same destination. The RREQ (A, ID_A) terminates when either node A attains a feasible sequence

number for destination D or the timer for its RREQ expires.

If node A is active or engaged for destination D and receives data packets for the destination, it buffers those data packets. If node A is passive for destination D and requires a route for destination D, it sets $ID_A \leftarrow$ *incremented request counter*, $reqid \leftarrow ID_A$, $msn \leftarrow sn_D^A$, $wc \leftarrow 1$, and *RREQ timer* $\leftarrow$ *(*2.ttl.latency) (where ttl is the time-to-live of the broadcast flood and latency is the estimated per-hop latency of the network). If $sn_D^A = -1$, then $reset = 1$. Node A then issues RREQ $\{D,\ A,\ reqid = ID_A,\ msn,\ wc, reset\}$.

If node A receives no RREP after the expiry of its timer for RREQ (A, ID_A) for destination D, it sends a new RREQ with an increased ttl. If node A does not receive a RREP for destination D after a number of attempts, a failure is reported to the upper layer. The number of hops that a RREQ can traverse is controlled externally from the RREQ by means of the TTL field of the IP packet in which a RREQ is encapsulated, or by other means.

4.3 Relaying RREQs

When node B receives a RREQ $\{D,\ \ A,\ \ rreqid = ID_A,\ \ msn_D^{req},\ \ wc_D^{req}, reset\}$ from node I, it first determines its own status for (A, ID_A). If B is active (i.e., $B = A$) or engaged (i.e., B has cached the RREQ (A, ID_A)) in the computation (A, ID_A), it silently drops the RREQ. Otherwise, node B is passive. In this case, if SSC is satisfied (i.e., $sn_D^B > msn_D^{req}$ and $reset = 0$), then node B issues a RREP (Section 4.4). Else, if SSC is not satisfied, node B becomes engaged and relays a new RREQ req' with the following parameter values: $msn_D^{req'} \leftarrow max\{sn_D^B,\ msn_D^{req}\}$, if $sn_D^B > sn_D^{req}$ then $wc^{req'} \leftarrow 1$ else $wc^{req'} \leftarrow wc^{req} + 1$, if $sn_D^A = -1$ then $reset_D^{req'} \leftarrow 1$ else $reset_D^{req'} \leftarrow reset_D^{req}$.

A node may be engaged in multiple RREQs for the same destination, but relays a RREQ from the same origin only once by caching the tuple $\{msn_D^{req},\ wc^{req'},\ I,\ reset_D^{req}\}$ for a given RREQ (A, ID_A) it forwards.

4.4 Initiating and Processing RREPs

When node I processes a RREQ $\{D, A, rreqid = ID_A, msn_D^{req}, wc_D^{req}, reset\}$ and SSC is satisfied (i.e., $sn_D^I > msn_D^{req}$, $rt_D^A = valid$, and $reset = 0$), it issues a RREP $\{D,sn_D^{rep}, A, ID_A, d_D^{rep},ttl,reset\}$ with $sn_D^{rep} \leftarrow sn_D^I$ and $d_D^{rep} \leftarrow d_D^I$. At the destination ($I = D$), if $reset_D^{req} = 1$, then D sets $sn_D^D \leftarrow sn_D^D + 1$; otherwise, if $sn_D^D \leq sn_D^{rep}$ then D sets, $sn_D^D \leftarrow sn_D^{req} + dstSeqInc$. If $reset_D^{req} = 1$, then D issues RREP with $reset = 1$. Section 4.6 addresses how the value for $dstSeqInc$ is chosen.

If node A receives a RREP $\{D,\ src = A,\ rreqid = ID_A,\ sn_D^{rep},\ ttl,\ d_D^{rep},reset\}$, it determines if it is the source src of the RREQ that caused the RREP. If so, it proceeds as Section 4.5 describes. If $A \neq S$, then after updating its routing table as per Section 4.5, the RREP is relayed along the reverse hop $revHop$, which is retrieved from the cache entry (A, ID_A) ; the RREP is relayed with $sn_D^{rep} \leftarrow sn_D^I$; $d_D^{rep} \leftarrow d_D^A$.

4.5 Adding, Updating, and Maintaining Routes

Node A updates its routing information when it receives a RREP $\{D, A,ID_A, sn_D^{rep}, ttl, d_D^{rep},reset\}$ from neighbor B. If $sn_D^A = -1$ and $reset_D^{rep} = 0$, or $reset_{(A,ID_A)}^{rep} = 1$ and $reset_D^{rep} = 0$, or if SNC is not satisfied, then the RREP is dropped silently. Node A calculates sn_D^{adj}, retrieving the values of $msn_{(A,ID_A)}^{rep}$ and $wc_{(A,ID_A)}^{rep}$ from the cache for the relayed RREQ (A, ID_A).

$$sn_D^{adj} = sn_D^{rep} - \lfloor \frac{sn_D^{rep} - msn_{(A,ID_A)}^{rep}}{wc_{(A,ID_A)}^{rep} + 1} \rfloor \tag{1}$$

Node A updates its routing table entry for D according to its current state(rt_D^A) as follows:

Case (a): If $rt_D^A = \phi\ \vee\ reset_{(A,ID_A)}^{rep} = 1$, then

$$sn_D^A \leftarrow sn_D^{rep} \tag{2}$$

For cases (b) and (c), $reset_{(A,ID_A)}^{rep}$ must be 0.

Case (b): if $rt_D^A = invalid\ \wedge\ sn_D^{rep} > sn_D^A$,

$$sn_D^A \leftarrow \begin{cases} sn_D^{adj} & \text{if } msn_{(A,ID_A)}^{rep} \geq sn_D^A \\ sn_D^A + 1 & \text{otherwise} \end{cases} \tag{3}$$

Case (c): if $rt_D^A = valid$,

$$sn_D^A \leftarrow \begin{cases} sn_D^{adj} & \text{if } msn_{(A,ID_A)}^{rep} \geq sn_D^A\ \wedge\ sn_D^{rep} > sn_D^A \\ sn_D^A & \text{if } msn_{(A,ID_A)}^{rep} < sn_D^A\ \wedge \\ & d_D^A > d_D^{rep} + lc_B^A\ \wedge\ sn_D^{rep} \geq sn_D^A \end{cases} \tag{4}$$

If the route reply was used to update the routing table sequence number entry, then node A sets $s_D^A \leftarrow B$; $d_D^A \leftarrow d_D^{rep} + lc_B^A$.

SWR handles link failures, and route lifetimes in the same way as AODV. However, a route table entry for a destination can be purged at any time. RERRs do not carry the destination sequence number in SWR.

4.6 Destination Sequence Numbers

To handle reboots and node failures, SWR uses a 64-bit destination sequence number based on a real-time clock which ensures that RREPs issued by the destination have

a non-decreasing sequence number. Additionally, after a reboot, a node will lose its cached state and the last-used flooding identifier for the RREQs. Hence, the *flooding identifier* (rreqid) carried in the RREQs must also be based on a real-time clock, because old flooding identifiers will not be relayed by nodes which previously processed a $(src, rreqid)$. The $rreqid$ can be truncated to a 32-bit integer, provided it will not wrap-around during for the time an old $(src, rreqid)$ might still be present in the network.

A RREQ that cannot be answered by any intermediate node will eventually reach the destination for a sequence number reset. AODV resets the destination sequence number with a higher sequence number than the one carried in the RREQ. To avoid the destination being the only node that can answer, SWR resets the destination sequence number by a parameter $dstSeqInc$. A linear-increment scheme with a pre-configured $dstSeqInc$ parameter should suffice for most network configurations. However, performance can be improved by using adaptive increment schemes which derive $dstSeqInc$ as a function of the prevailing network conditions (i.e., number of RREQs received within a time interval).

4.7 Reverse Routes

A RREQ generated by a source can be considered as a RREP in the reverse direction. However, SNWs cannot be used for setting up routes in the reverse direction, because of the lack of window boundaries. Hence, SWR uses an optimization to use SNWs. If node A receives a RREQ from B and has no valid route towards the source S of the RREQ, node A performs the following steps: creates a new route entry for the source S, sets $s_S^A \leftarrow B$, sets the lifetime of the route equal to the *reverse route lifetime*, and flags the route entry with a special bit indicating that it is an invalid reverse route. When node A has a flagged reverse route to S needs to send data packets to that destination, it sends a unicast RREQ to s_S^A. This RREQ is forwarded on a hop-by-hop basis along a path of nodes with invalid reverse routes to S, and a RREP can be generated by either a node satisfying SSC or the destination. A unicast RREQ follows the same rules as a broadcast RREQ.

5 SWR Example

Fig.4 shows a directed acyclic successor graph (DASG) for destination D for a six-node network at time t_1. Initially at time t_0, the routes are not established, the destination D has a un-initialized sequence number (0); and other nodes in the network do not possess any knowledge about D. Node A initiates a route request req for destination D with parameters (D,A,$rreqid = ID_A$,$sn_D^A = -1$,$wc = 1$,$reset = 1$). Node B upon receiving the route request (A, ID_A), caches

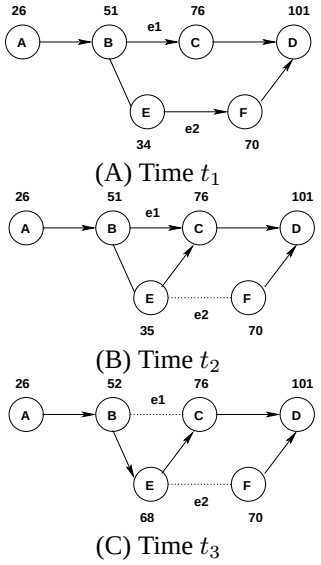


Figure 4. SWR operation - Example

the tuple ($msn_D^{req} = -1$,$wc = 2$,$revHop = A$, $reset = 1$), and relays a new RREQ req with (D,A,$rreqid$,$sn_D^{req} = -1$,$wc = 2$,$reset = 1$). Similarly, node C relays the RREQ. Note that there is no significance for wc, when a reset is requested.

A route reply rep is generated by node D upon receipt of the RREQ by D. Node D increments its destination sequence number by just one, because it is performing a reset. The RREP (D,A,ID_A,$sn_D^{rep} = 1$,$reset = 1$) is accepted by node C as it satisfies ASC . By Eq. 2, node C sets $s_D^C \leftarrow D$; $sn_D^C \leftarrow 1$; and relays the RREP along the cached reverse hop. Similarly, since the RREP carries $reset = 1$, nodes A and B have to update their destination sequence numbers to one. Now, all nodes possess knowledge of D's sequence number. The above additional steps are necessary to ensure the correctness of the protocol. To illustrate the use of SNWs, assume that the nodes expire their route lifetime; they have marked their route entry for D as invalid.

Now, assume node A starts another RREQ $(D, A, ID'_A, sn_D^A = 1, wc = 1, reset = 0)$. Nodes B and C relay the RREQ after increasing the wc to two and three, respectively. Node D, on receiving the RREQ, now increments its sequence number to 101, using a linear increment (say $dstSeqInc = 100$). Node C accepts the RREP because it satisfies ASC, adds a new routing table entry for destination D, sets $s_D^C \leftarrow D$, and calculates a sequence number of 76 for destination D using Eq. 3 (i.e., $101 - \lfloor \frac{(101-1)}{3+1} \rfloor$). Similary, node B and node A update their routing tables to setup an entry for destination D with sequence numbers, 51 and 26, respectively, using Eq.3. At time t_1, there is a progressive ordering of destination sequence numbers from node A to destination D.

After a similar sequence of events, assume that nodes E and F have set their destination sequence number for D to 34 and 70, respectively, at some time $t < t_1$. At time $t_2 > t_1$, link e_2 fails. Node E detects a link failure and sends the RREQ (D,E,ID_E,$msn_D^{req} = 34$,$wc = 1$) which evokes a response from C which satisfies SSC. Node E activates routing entry for D after processing the reply and updating $sn_D^E \leftarrow 35$ as per Eq. 3. Fig. 4(b) shows the state of the network.

Let node B detect the failure of link e_1 and sends the RREQ (D,B,ID_B,$msn_D^{req} = 51$,$wc = 1$) at a later time $t_3 > t_2$. Node E does not satisfy SSC and relays a new RREQ (D,B,ID_B, $msn_D^{req} = 51$,$wc = 2$). Note that there is a window between B and C spanning node E. Node C responds to the RREQ with a RREP carrying $sn_D^{rep} = 76$, because SSC is satisfied. Node E processes the RREP and sets $sn_D^E \leftarrow 68$ as per Eq. 4, which amounts to redistributing sequence numbers in the window between B and C. Node B re-establishes a route to D after updating its route entry to set $sn_D^B \leftarrow 52$ and $s_D^B \leftarrow E$. In this case, we assume that node B is performing a localized route repair, without which node A would have received a RERR from B and initiated a route search for D. Fig. 4(c) shows the directed acyclic successor graph at time t_3.

This example illustrates that the progressive ordering of sequence numbers allows SWR to recover quickly from route failures. AODV operating in this scenario would have required a sequence number reset from the destination.

6. Analysis

For SWR to be loop-free, it must be true that any successor path to a destination must have monotonically non-decreasing sequence numbers, even when nodes forget their sequence numbers for a destination, and loop-freedom follows directly from the SNC proof of loop-freedom [7].

Theorem 2 *If a node updates its routing table as per Section 4.5, then the sequence number towards a destination at a node is a non-decreasing function of time, even if nodes lose their sequence-number state.*

Proof: The route entry for destination D at node A can be in one of the three states when it must update its routing-table entry after receiving a RREP rep: (i) no information, (ii) invalid , or (iii) valid.

In case (i), the route entry does not exist ($sn_D^A = -1$), and the RREP will be accepted only if $reset = 1$. Since the destination sequence number is incremented by atleast 1 when a RREQ with $reset = 1$ is answered, the RREP must carry a sequence number which must be higher than any previously known to node n_i; and it updates itself to sn_D^{rep}.

For the other cases, we first derive the range of sn_D^{adj} calculated from Eq.1 when $msn_{(src,rreqid)}^{rep} \geq sn_D^A$, assuming $sn_D^{rep} > sn_D^A$. Say $sn_{(src,rreqid)}^{rep} = sn_D^A + \alpha$ and $sn_D^{rep} = sn_D^A + \beta$, where α, β are integers such that $\alpha \geq 0$ and $\beta > 0$. Then from Eq.1 , we have $sn_D^{adj} = sn_D^A + \beta - \lfloor \frac{\beta - \alpha}{\lambda} \rfloor$, where $\lambda = wc_{(src,rreqid)}^{rep} + 1 > 1$, and $\beta > \alpha$. Therefore , we have the inequality

$$sn_D^A \leq msn_{(src,rreqid)}^{rep} < sn_D^{adj} \leq sn_D^{rep} \tag{5}$$

In case (ii), from Eq. 3, either sn_D^A is set to sn_D^{adj} or incremented by one ($sn_D^A + 1$), and the sequence number only increases in both cases. In case (iii), from Eq. 4, the sequence number (sn_D^A) increases to the new adjusted value (sn_D^{adj}), or remains the same when the distance gets shorter.

If on the other hand, reset =1 in the cache, and the node falls under any of the above three cases: The update follows Eq.2 which ensures that the node updates to a higher sequence number, using the same argument as in case (i).

Hence the sequence number for the destination is non-decreasing when the node accepts and updates its routing table as per Section 4.5. ■

Theorem 3 *At any instant, SWR is loop-free.*

Proof: The proof for loop-freedom of SWR follows directly from Theorem 1 and 2, because ASC is equivalent to SNC, provided that we can show that SLC is satisfied when route-table entries are updated.

Consider a node n_i having a route to destination D. The predecessor of n_i, denoted by n_{i-1}, can make n_i its next hop towards D only if SNC is satisfied. Now, node n_{i-1} cannot increase its sequence number without receiving a new update from n_i. From Theorem 2, node n_i can only increase or maintain the same sequence number (path is of shorter cost which is not a SLC case) when it switches to a new successor, which means that SLC is satisfied, i.e., $sn_D^{n_i} > sn_D^{n_i - 1}$, on a sequence number update. At a later time, node n_i might relay another RREP to a new predecessor m_i, but following the same argument it can only increase its sequence number before relaying the RREP, which ensures that $sn_D^{n_i} > sn_D^{n_i - 1}$, or if node n_{i-1} increases its sequence number, without receiving any new update from node n_i, it can only do so because it switched to another successor. ■

To prove the correct termination of SWR, we first show that any source is able to establish a route to a destination within a finite time if there is a physical path between the source and the destination, assuming that the network is stable and error-free after an arbitrary sequence of topology changes. The proof is similar to the convergence proof of LDR considering only sequence numbers [2](Theorem 5, pp.60).

We give only an outline of the proof: Let source A issue a RREQ req which traverses a path, $P = \{n_1, n_2, ..., n_i\}$, before reaching the destination or a node that satisfies SSC. The RREP issued will have a sequence number greater than msn_D^{req}, which will satisfy ASC along the entire path and at A. If A or a node along path P requested a reset, then the RREP will carry $reset = 1$, which will satisfy any such node. Because the RREP is now relayed along the reverse path, the RREP must satisfy ASC, given that the relaying nodes after updating their routing tables follow MSC, and the RREP at the relaying node has $sn_D^{rep} > msn_{(A,ID_A)}^{req}$ acording to Eq. 5. When the RREP traverses the reverse path, if it does not satisfy ASC at a node, then the node learned a route with a higher sequence number and the new RREP generated will still satisfy ASC at the nodes along the reverse path to the source. In any case, the RREP forwarded along the reverse path will satisfy all the nodes and hence the source will be able to establish a successor path in finite time due to finite time for message exchanges.

Next, we show that all nodes invalidate their routing table entries for the destination in the presence of link failures and node reboots/state loss that disconnect some nodes from a destination. Following the default RERR rules, the RERRs should eventually propagate upstream along the acyclic successor graph in finite time. During this time, we argue that nodes cannot keep learning newer routes from upstream nodes, which can lead to count-to-infinity behavior. When nodes reboot or lose state, only RREPs with $reset = 1$ can be used to update their route entries, which is not possible in a partitioned network and hence these nodes can never learn a new route. On link failures or otherwise, nodes with valid sequence number entries can learn routes from a node that has a higher sequence number than the one in the request. However, because only the destination can reset (increase) its destination sequence number, within a finite time all nodes should have the highest destination sequence number entry and future route searches cannot be answered by any node in this partition. Assuming a finite probability that RERR messages will eventually be delivered, all nodes will invalidate their route entries for the destination.

7 Performance

We present results for SWR over varying loads and mobility. The protocols used for comparison are two on demand protocols DSR and AODV, which reflect the state of the art in on-demand routing, and OLSR which is a proactive link state protocol. Simulations are run in Qualnet 3.5.2. The parameters are set as in [9].

Simulations are performed on two scenarios, (i) a 50-node network with terrain dimensions of 1500m x 300m, and (ii) a 100-node network with terrain dimensions of 2200m x 600m. Traffic loads are CBR sources with a data packet size of 512 bytes. Load is varied by using 10 flows (at 4 packets per second) and 30 flows (at 4 packets per second). The MAC layer used is 802.11 with a transmission range of 275m and throughput 2 Mbps. The simulation is run for 900 seconds. Node velocity is set between 1 m/s and 20 m/s. Flows have a mean length of 100 seconds, distributed exponentially. Each combination (number of nodes, traffic flows, scenario, routing protocol and pause time) is repeated for nine trials using different random seeds.

We present four metrics. *Delivery ratio* is the ratio of the packets delivered per client/server CBR flow. *Latency* is the end to end delay measured for the data packets reaching the server from the client. *Network load* is the total number of control packets (RREQ, RREP, RERR, Hello, TC etc) divided by the received data packets. *Data hops* is the number of hops traversed by each data packet (including initiating and forwarding) divided by the total received packets in the network. This metric takes into account packets dropped due to forwarding along incorrect paths. A larger value for the data-hops metric indicates that more data packets traverse more hops without reaching the destination necessarily.

Tables 1, and 2 summarize the results of the different metrics by averaging over all pause times for the 50-node and 100 -node networks. The columns show the mean value and 95% confidence interval. Fig. 5 shows the delivery ratio for a 100-node network with 30-flows. Confidence intervals(95%) are shown with vertical bars in the graphs.

SWR has a very consistent performance across all scenarios and outperforms other protocols in most cases. In the highest load scenario (100 nodes, 30-flows), SWR has the highest packet delivery (0.6952 ± 0.04), and the lowest latency (0.9211 ± 0.17). The exception is at high flows, low mobility scenarios where OLSR seems to do better. The latency of SWR is more than that of AODV in the 10-flow scenarios; but that is due to the overhead incurred by the additional mechanisms for correctness of the protocol. In the 100-node, 30-flow scenarios, we believe that AODV suffers from convergence problems as indicated by the poor confidence intervals obtained for the control overhead(18.29 ± 13.06). The performance of DSR suffers from stale caches, as indicated by its low packet delivery and high latency; and using cached source routes avoids route requests floods, characterized by the low control overheads. OLSR performs well in the low mobility scenarios; but, on the whole, is affected by poor performance in the high-mobility scenarios.

The *data hops* metric provides a measure of the accuracy of the routes used for forwarding. SWR, AODV and OLSR have statistically equivalent data hops across all scenarios, except in the scenario with 30 flows and 100 nodes where

Table 1. Performance average over all pause times for 50 nodes network for 10-flows and 30-flows

Flows	10	30	10	30	10	30	10	30
Protocol	Delivery Ratio		Latency (sec)		Net Load		Data Hops	
SWR	0.995±0.001	0.826±0.046	0.020±0.002	0.658±0.250	0.338±0.080	2.858±0.871	2.582±0.180	2.818±0.283
AODV	0.994±0.002	0.765±0.0553	0.016±0.003	1.010±0.356	0.270±0.066	4.423±1.289	2.576±0.179	2.951±0.324
DSR	0.940±0.027	0.683±0.059	0.041±0.047	4.760±1.073	0.220±0.095	0.410±0.140	2.677±0.185	3.625±0.308
OLSR	0.887±0.040	0.798±0.034	0.012±0.001	0.883±0.311	1.937±0.220	0.713±0.069	2.456±0.175	2.478±0.161

Table 2. Performance average over all pause times for 100 nodes network for 10-flows and 30-flows

Flows	10	30	10	30	10	30	10	30
Protocol	Delivery Ratio		Latency (sec)		Net Load		Data Hops	
SWR	0.989±0.004	0.695±0.045	0.053±0.010	0.921±0.174	1.423±0.402	10.027±1.800	3.757±0.317	4.368±0.353
AODV	0.988±0.004	0.608±0.051	0.036±0.009	1.455±0.385	0.897±0.236	18.298±13.069	3.744±0.293	4.751±0.434
DSR	0.876±0.050	0.618±0.049	0.099±0.057	5.125±0.782	0.859±0.353	1.243±0.405	4.257±0.317	6.141±0.499
OLSR	0.821±0.063	0.612±0.041	0.022±0.002	3.371±0.532	11.795±1.575	5.423±0.669	3.583±0.256	4.014±0.277

DSR incurs max data hops. The data hops metric reflects the number of hops traversed by each data packet whether or not it is delivered. SWR's data hops in correlation with the packet delivery ratio shows the high accuracy of the active routes.

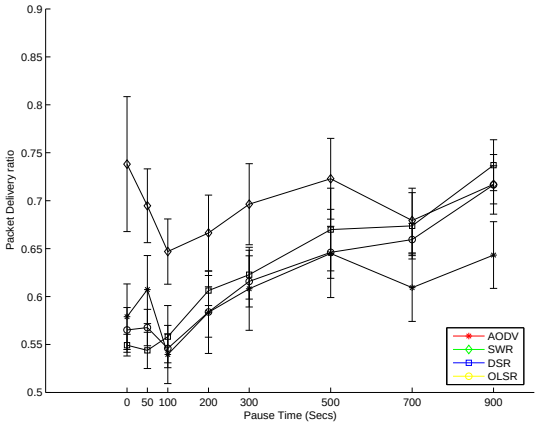


Figure 5. Delivery 100-nodes, 30-flow, 120pps

8 Conclusion

We extended the loop-free conditions for destination sequence number based protocols to allow nodes to treat sequence numbers as labels instead of absolute timestamps. We introduced a generic framework for on-demand protocols based on destination sequence number based protocols that is safe even when nodes are allowed to forget previously learned sequence numbers. We presented the Sequence Window Routing (SWR) protocol which adds sequence number windows to the framework and progressively labels the sequence numbers towards a destination. This novel scheme maintains loop-free routes while at the same time reducing control overhead and latency for setting up routes improving overall packet delivery. Simulation results show that SWR provides comparable performance than that of AODV while eliminating counting-to-infinity and looping problems when nodes can forget the sequence numbers to destinations.

References

[1] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol," Request for Comments 3626, October 2003.

[2] J. J. Garcia-Luna-Aceves, M. Mosko and C. Perkins, "A New Approach to On-Demand Loop-Free Routing in Ad Hoc Networks," *Proc. ACM PODC 2003*, pp. 53-62, Boston, Massachusetts, July 13–16, 2003.

[3] J. J. Garcia-Luna-Aceves and M. Spohn, "Source-Tree Routing in Wireless Networks," *Proc. IEEE ICNP'99*, pp. 273–82, Toronto, Canada, October 31–November 3, 1999.

[4] D. Johnson et al, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)," IETF Internet draft, draft-ietf-manet-dsr-09.txt, April 2003.

[5] R. Ogier et al., "Topology Dissemination Based on Reverse-Path Forwarding (TBRPF),"Request for Comments 3684, February 2004.

[6] H. Rangarajan and J.J. Garcia-Luna-Aceves, "Using Labeled Paths for Loop-free On-Demand Routing in Ad Hoc Networks," *Proc. ACM MobiHoc 2004*, Tokyo, Japan, May 24–26, 2004.

[7] C. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," *Proc. ACM SIGCOMM 94*, 1994.

[8] C. Perkins et al., "Ad hoc On-Demand Distance Vector (AODV) Routing," Request for Comments 3561, July 2003.

[9] C. Perkins et al. "Performance Comparison of Two On-demand Routing Protocols for Ad Hoc Networks," *IEEE Personal Communications*, 8(1):16 – 28, Feb 2001.